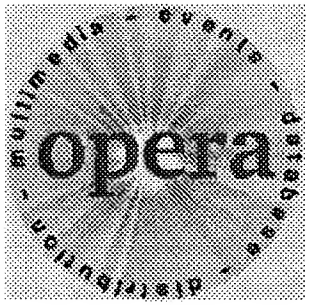
JavaBeans

Component Architecture for Java

UNIVERSITY OF
CAMBRIDGE
Computer Laboratory



Alan Abrahams

alan.abrahams@cl.cam.ac.uk

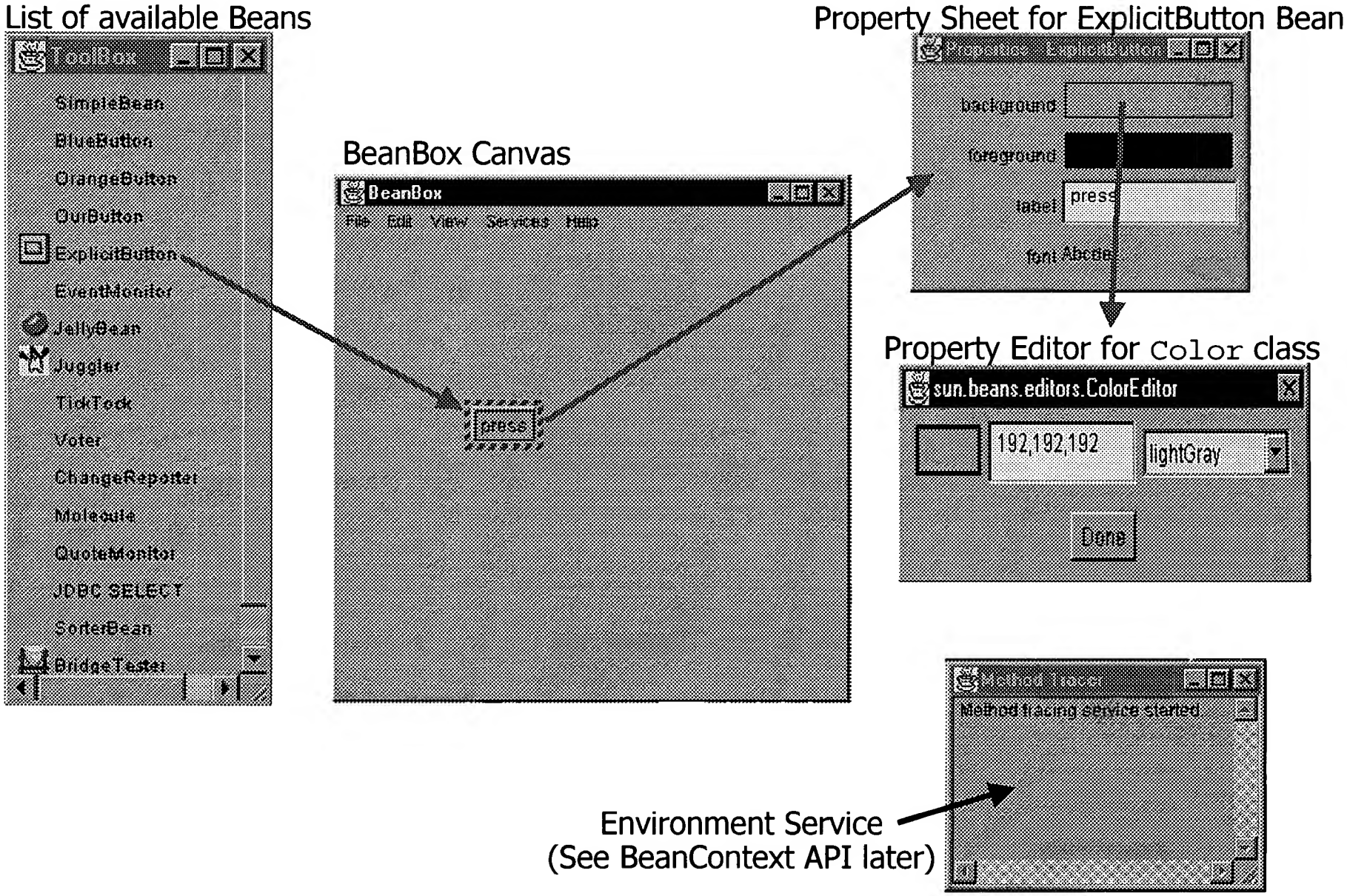
Overview

- What are Beans?
- The BeanBox
- Reflection and Introspection
- JavaBeans 'Features'/Capabilities:
  - Properties, Methods, Events, Persistence
- Packaging
- Customization
- Enterprise JavaBeans
- Recent Enhancements to JavaBeans
  - BeanContext API, Drag and Drop, JavaBeans Activation Framework (JAF), and the InfoBus

What are **Beans**?

- Portable, re-usable, self-contained, visual or non-visual components that can be manipulated visually in a builder tool, and visually composed/assembled.
- Examples: ButtonBean, FTPBean, EventMonitorBean
- Regular Java classes that:
  - Follow naming conventions (Referred to incorrectly as 'design patterns' in JavaBeans spec) to expose their 'features' (i.e. properties, methods, & events)
  - May use supplementary classes to customize visibility of 'features' (`BeanInfo`) or editability (`Customizer` classes)
  - Can persist (i.e. implement `Serializable` or `Externalizable`)
  - Implement a zero-argument constructor

The BeanBox


List of available Beans
ToolBox
SimpleBean
BlueButton
OrangeButton
OurButton
ExplicitButton
EventMonitor
JellyBean
Juggler
TickTock
Voter
ChangeReporter
Molecule
QuoteMonitor
JDBC SELECT
SorterBean
BridgeTester
BeanBox Canvas
BeanBox
File Edit View Services Help
press
Property Sheet for ExplicitButton Bean
background
foreground
label
press
font Abcde
Property Editor for Color class
sun.beans.editors.ColorEditor
192,192,192
lightGray
Done
Environment Service
(See BeanContext API later)
Method tracing service started

Reflection and **Introspection**

- Java's **Reflection** API allows classes to 'look inside' other classes
- **Introspection** is built on top of reflection, and enables a builder tool to determine a Beans 'features' (properties, methods, and events). Introspection uses reflection but adds:
  - 'Design Patterns' (or rather, 'naming conventions')
  - The `BeanInfo` class to explicitly describe Bean 'features' or provide different 'views' of the bean. An appropriately-named `BeanInfo` class will override defaults and change property, method, and event visibility for the similarly-named Bean. e.g. by occluding methods or providing more understandable synonyms for method or property names.

JavaBeans 'Features'

- JavaBeans 'Features':
  - Properties
  - Methods
  - Events
  - Persistence

Properties

- Properties are named attributes that can be read or written by calling appropriate methods on the Bean.
- There are four kinds of Properties:
  - **Regular:** Properties that are objects of a particular type (class or interface)
  - **Indexed**: Properties that are arrays
  - **Bound**: Other beans are notified of property changes through PropertyChangeEvents.
  - **Constrained**: Other beans are notified of changes about to occur and can veto these changes. Vetoes appear to be synchronous.

Methods

- These are normal Java methods.
- By default all the `public` methods of the Bean are exported (i.e. externally visible and invokable). These `public` methods are sometimes called **control** methods as they can be used to manipulate or instruct the bean.
- An appropriately-named `BeanInfo` class can be used to export only a sub-set of these methods (i.e. to 'occlude' certain methods).

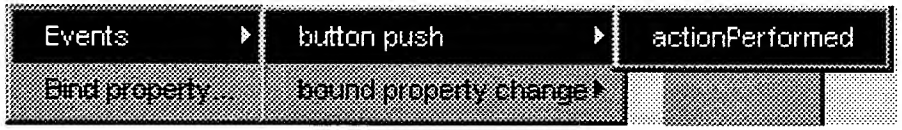
Events

- Beans communicate by sending and receiving event notifications (`EventObject`s).
- Events are implicitly **publish**ed by event sources by following naming conventions, or explicitly published through a supplementary `BeanInfo` class. The fireable events for a bean are determined through Introspection.
- Event Listeners **register** with event sources
- Event sources (JavaBeans) **notify** Event Listeners (sinks).
- Event Listeners invoke `public` methods (a.k.a. **control** methods) of target Bean.

Runtime Event Hookup

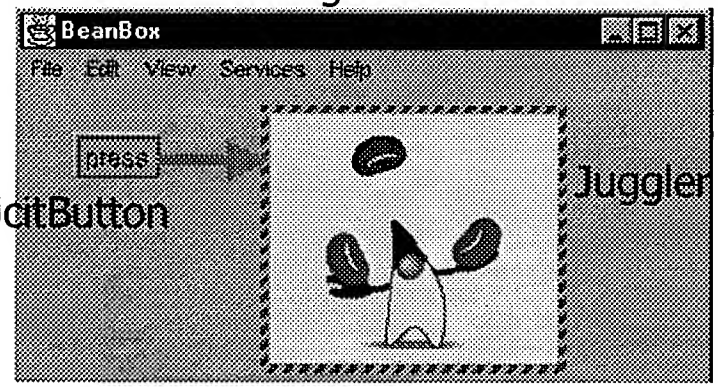
- Adapter/Hookup classes can be generated at run-time to tie events to `public` (i.e. control) methods dynamically.

Events exposed by ExplicitButton Bean



Public methods (i.e. control methods) exposed by Juggler Bean

Dynamically-generated Hookup class which implements the appropriate EventListener, and passes the event on to the selected public method of the target bean.



ExplicitButton

Dynamically-generated Hookup class (which implements the appropriate EventListener) must be registered at the event source.

Persistence

- Beans can be serialized to save and restore their **state** (not their behaviour/methods, which are stored in the class file for the Bean).
- Event hookups/adaptors (i.e. bean 'wiring') can also persist, and the hookup classes are class files.
- So Beans and **assemblies** of Beans can be customised and then saved for future use.
- Fields can be marked `transient` or `static` if their values are not required to persist.
- Serialization also allows **transmission** of bean state **over the network**.
  Remember, Bean behaviour is not transmitted so the relevant Bean class needs to be available to the class loader at the remote location.
- **Versioning** (through version numbers) is provided by default, but the strict default versioning scheme can be overriden.

Packaging

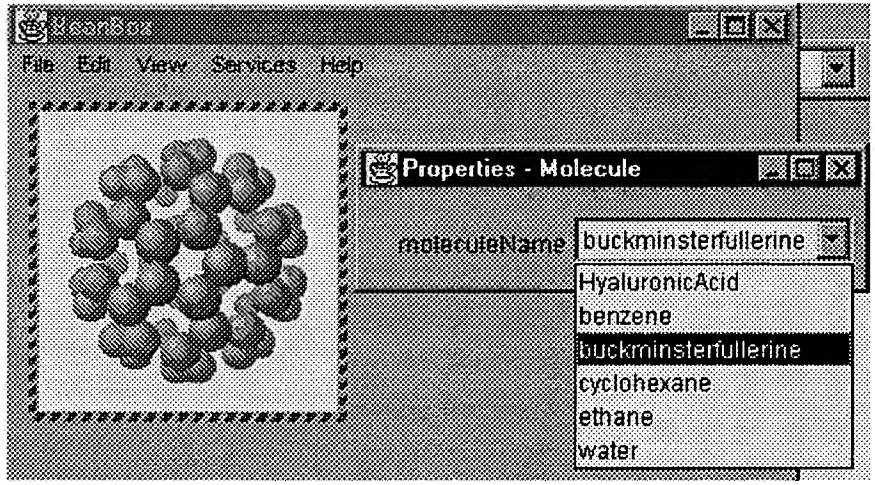
- Beans (i.e bean classes) and related `BeanInfo` classes, `Customizer` classes, and resources (e.g. images, audio, text messages, serialized objects such as a serialized prototype bean used to initialize the bean, MIME encoded data, optional help files in HTML format) are packaged together into **JAR (Java Archive) files.**
- JAR files are effectively zip files with plain-text manifests that describe their contents.
  (The format of the manifest is described in the JavaBeans Specification [JAV96])

Customization

- Altering Bean appearance, Bean behaviour, and Bean manipulation.
- Customization occurs at various levels
  1. Manipulate Bean properties (e.g. set a `Button`'s `label` at run-time, or invoke the Bean's property `set`ter methods). Properties must have been exposed for Introspection either implicitly (through use of JavaBeans naming conventions) or explicitly (through an appropriately-named `BeanInfo` class)

Customization
(...cont)

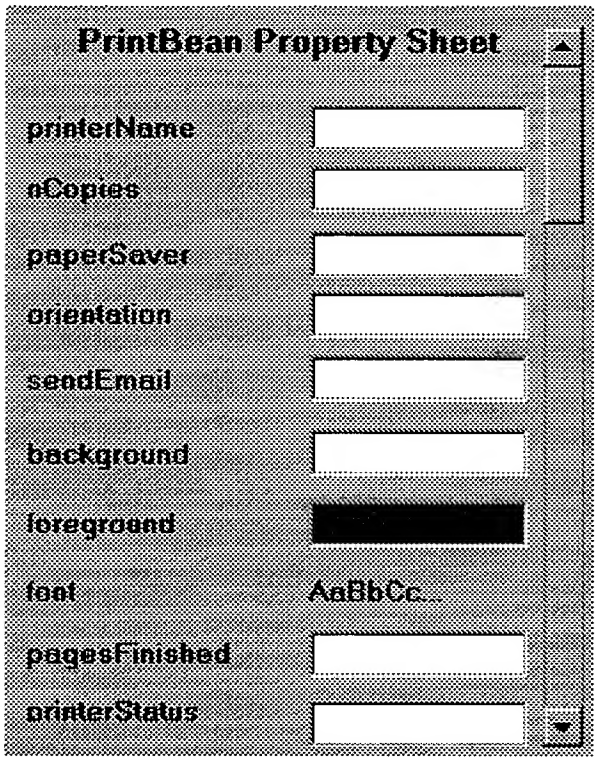
2. Provide `PropertyEditors`. BeanBox provides default property editors for the primitive types, plus `Font` and `Color` classes. Custom editors can be created for other property types (classes).



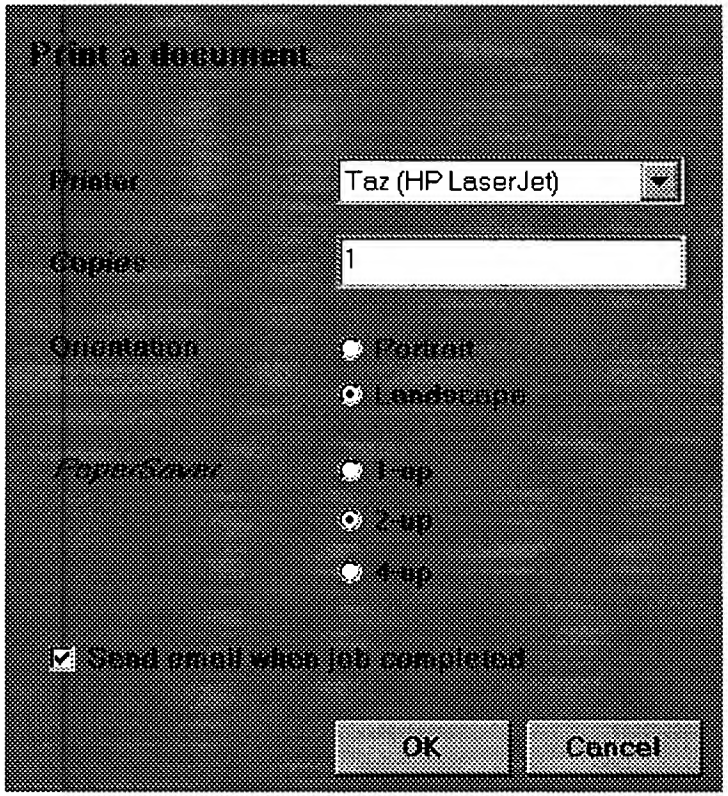
The Molecule Bean and a custom PropertyEditor for the moleculeName property

Customization (...cont)

3. Provide Customizers which implement the `Customizer` interface.



Default Property Sheet



Customizer

Source:

Johnson M: *The trick to controlling bean customization*, JavaWorld, November 1997, Accessed at: http://www.javaworld.com/javaworld/jw-11-1997/jw-11-beans_p.html

Enterprise JavaBeans (EJB)

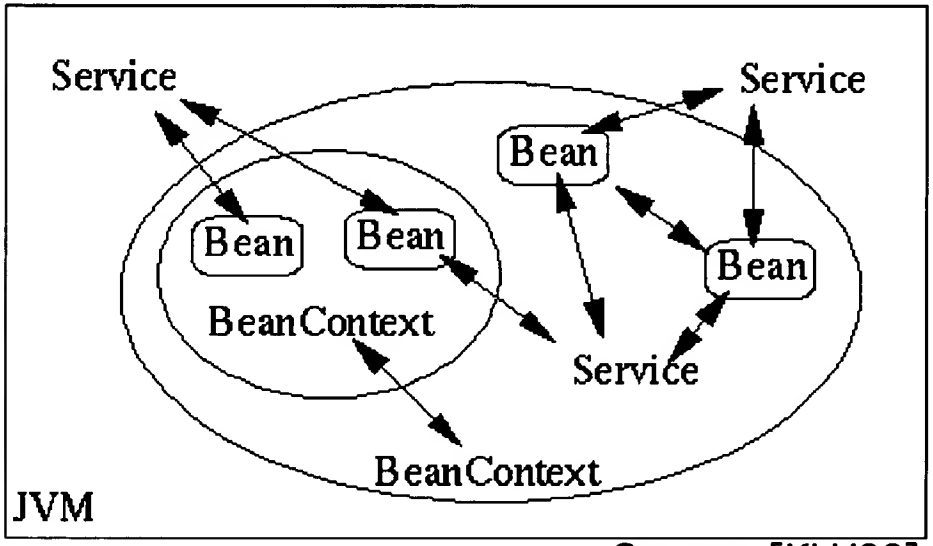
- EJB is used for building scalable, distributed, component-based, multi-tier applications.
- Extends JavaBeans to middle-tier and server-side business applications.
- Uses Session and Entity Beans, Containers, etc.

Perhaps the topic of a future presentation...

BeanContext API

(Extensible Runtime Containment and Services Protocol)

- Allows nested Beans
- Allows a Bean to discover **attributes** and **services** in its environment (surrounding `BeanContext`) at run-time.
- Allows the `BeanContext` to manipulate the Beans it contains, and force settings on such beans.



Source: [KLU99]

Drag and Drop

- Support for native platform drag and drop gestures
  - *Transferable* data is dragged from a *DragSource* to a *DropTarget*.
- Uses the existing `java.awt.datatransfer.*` package to enable the transfer of data described by an extensible data type system based on the MIME standard.

JavaBeans Activation Framework (**JAF**)

JAF provides a standard set of services that make it possible to:

- Determine the type of an arbitrary piece of data
- Encapsulate access to that data
- Discover the operations available on that data
- Instantiate the appropriate JavaBeans components to perform said operations (in order to view, edit, translate/convert, or print the data).

(e.g. determine MIME type such as JPEG and instantiate and image viewer or editor.)

InfoBus

- Allows JavaBeans attached to an InfoBus (executing within a single JVM) to exchange data asynchronously, by data item name
- Data Producers, Data Consumers, Data Controllers
- To communicate with Beans executing in other JVM's (i.e. to access distributed data), Data Producers must be used to obtain the data (e.g. via SQL, HTTP, XML, CORBA, RMI, etc.).
- Policy Helpers provide a rudimentary interface that allows control of who can join an InfoBus, and who can request, provide, and/or consume data.

Interesting Technologies

- IBM AlphaWork's Beans Markup Language: To declaratively compose beans into applications using XML-based BML.
- IBM Alphawork's XML Beans Suite: JavaBeans that can be interconnected visually to read, write, display, search, and filter XML data.
  See also www.xbeans.org for a similar JavaBeans technology for processing XML.

References

- **[Jav96]**
  JavaBeans 1.01 API Specification, JavaSoft
- **[CAB98a]**
  Cable L: Extensible Runtime Containment and Services Protocol v1.0, JavaSoft, 1998
- **[CAB98b]**
  Cable LPG: The Drag and Drop Subsystem for the Java Foundation Classes v0.96, JavaSoft, 24 August 1998
- **[COL99a]**
  Colan M: InfoBus 1.2 API Specification, Sun/Lotus Development Corp, 10 February 1999
- **[Jav99]**
  The JavaBeans Activation Framework v1.0.1, JavaSoft

Available from www.javasoft.com/beans/